OSTROLENK, FABER, GERB & SOFFEN, LLP

1180 AVENUE OF THE AMERICAS, NEW YORK, NEW YORK 10036-8403
TEL 212 382 0700 FAX 212 382 0888 FAX 212 398 0681
email@ostrolenk.com

PARTNERS

SAMUEL H. WEINER
ROBERT C. FABER
MAX MOSKOWITZ
JAMES A. FINDER
WILLIAM O. GRAY, III
LOUIS C. DUJMICH
CHARLES P. LAPOLLA
DOUGLAS A. MIRO
PETER S. SLOANE
KOUROSH SALEHI**

ASSOCIATES

JOEL J. FELBER**
MICHAEL I. MARKOWITZ
KEITH J. BARKAUS
JEFF KIRSHNER
ART C. CODY
DAVID J. TORRENTE
ANNA VISHEV
CAMERON S. REUBER***
ANGELA MARTUCCI

OF COUNSEL

MARTIN PFEFFER
MARTIN J. BERAN
PAUL GRANDINETTI*
MARK A. FARLEY
GEORGE BRIEGER
STEPHEN J. QUIGLEY
JOSEPH M. MANAK

WASHINGTON OFFICE
1120 20th STREET
SOUTH TOWER, SUITE 750
WASHINGTON, D.C. 20036
TEL 202 457 7785
FAX 202 429 8919

*DC BAR
**CONNECTICUT BAR
***VIRGINIA BAR

FACSIMILE TRANSMITTAL SHEET

DATE: January 8, 2009

NUMBER OF PAGES, INCLUDING COVER: 5

TO:

NAME/COMPANY	FACSIMILE NO.	
Mr. Isaac Tuku Tecklu U.S.P.T.O. Examiner	1-571-273-7957	☐ SUCCESSFULLY FAXED
		☐ SUCCESSFULLY FAXED
		☐ SUCCESSFULLY FAXED

FROM: George Brieger

OFGS FILE NO.: P/1905-108 RETURN TO: Norman Schiff

IF YOU DID NOT RECEIVE ALL THE PAGES, PLEASE PHONE (212) 382-0700 AS SOON AS POSSIBLE.

MESSAGE:

Attached please find proposed Interview Agenda with claim Appendix for a telephone interview scheduled for Thursday, January 8, 2009 at 4:00 p.m.

P/1905-108

DRAFT PROPOSAL FOR CLAIM AMENDMENT

Serial No.: 10/780,225
Examiner: Isaac Tuku Tecklu Facsimile No.: 1-571-273-7957
Group Art Unit: 2192
For: METHOD OF CONVERTING SOFTWARE PROGRAM FOR SINGLE PROCESSOR TO SOFTWARE PROGRAM FOR MULTIPROCESSOR
Date: January 8, 2009
Applicant's Representative: George Brieger, Reg. No. 52,652

PROPOSED AGENDA FOR EXAMINER INTERVIEW

NOT FOR THE PERMANENT RECORD OF THE USPTO.

EXAMINER EYES ONLY. PLEASE DISCARD AFTER THE INTERVIEW.

Claim Language

Claim 1 requires a method of converting a software program comprising object files for a single processor to a software program for a multiprocessor, the method comprising preparing an executable form program for each processor so that at the time of program execution by a respective processor of the at least two processors each executable form program is running on a single memory space, detecting an occurrence of a refer request to a variable arranged on a memory space managed by another processor of the at least two processors during running of the executable form program, sending the refer request to a requested processor of the at least two processors, and returning refer results, by the requested processor referring to the variable, to the refer requester processor.

Deficiencies of Martin

The Office Action alleges that Martin discloses "exception processing" or refer requests, citing Martin, column 10, lines 30-49 and the concurrent access to data discussed in that passage. However, as discussed previously, the exception processing required by claim 1 occurs "during running of the executable form program." That is, the method includes preparing an executable form program to be executed by each processor and exception processing during running of the executable form program that has been thus prepared.

By way of contrast, Martin is directed to a pre-compiler 316 that reads for each class the number of concurrent processes that may access objects of the class, but is silent with respect to any kind of exception processing performed after the preparation of an executable form program, as required by claim 1. Reading the number of concurrent processes that may access objects of a class does not constitute exception processing of an executable form program, as required by claim 1. Further, Martin is silent regarding any kind of exception processing performed during running of the executable form program, as further required by claim 1.

As discussed, Martin discloses a compiler or a pre-compiler which takes into account system data written into a comment field of the source code. This compiler or pre-compiler looks for such comment field information ignored by the conventional compiler. Thus, Martin is unconcerned with exception processing during run time of the executable form program, as required by claim 1.

Deficiencies of Berkovich

The Office Action newly cites Berkovich et al., U.S. Patent No. 5,619,680. Berkovich discloses concurrent execution of serial computing instructions using a combinatorial architecture for program partitioning so that applications developed for a single processor can be executed concurrently by allocating instructions of the program to various processors depending upon the addresses contained within operands of the instructions (Berkovich, Abstract). Thus, each processor (also referred to as a processing element (PE)) can execute instructions which inherently have all information needed in their registers at the time of execution (Berkovich, Abstract).

Berkovich discloses that, in addition to the processors, several memory control units or memory coordination units (MCUs) are provided, such that each MCU uniquely controls access to a particular block of memory (Berkovich, column 2, lines 41-48). Instructions of a program are then allocated to the PEs and the memory space of the system is partitioned into memory blocks such that each memory block is controlled by only one MCU (Berkovich, column 3, lines 7-11). Each PE is assigned to serve two or more MCUs (Berkovich, column 3, lines 11-13). The instructions of the program are allocated to a PE which serves the memory blocks containing the addresses of operands specified within those instructions (Berkovich, column 3, lines 13-15). As illustrated in Berkovich, Fig. 6, the instructions of the program are allocated such that an

instruction which would require access to data elements in memory sections 2 and 6 are assigned to PE 8 because PE 8 connects to MCU 2 and to MCU 6, and these two MCUs control the two areas of the memory space referenced by the instruction assigned to PE 8 (Berkovich, column 5, line 63 – column 6, line 4).

Berkovich does not disclose or suggest exception processing for a refer requestor processor of the at least two processors, by detecting an occurrence of a refer request to a variable arranged on a memory space managed by another processor of the at least two processors during running of the executable form program, and sending the refer request to a requested processor of the at least two processors, as required by claim 1. First, Berkovich does not disclose or suggest that any processor of the at least two processors, that is, the processors that execute executable form programs, manages a memory space, as required by claim 1. In fact, Berkovich discloses that another set of processors, the MCUs, manage and control access to each memory space. Thus, Berkovich actually teaches away from the recitations of claim 1 because Berkovich discloses introducing a class of processors known as MCUs to manage respective areas of memory. Further, Berkovich teaches away from Martin because Berkovich discloses the disadvantages of environments in which "source code is changed to include special symbols which indicate which operations are capable of being executed in parallel," or environments in which "certain directives are necessary to allocate instructions to specific ones of the parallel processors" (Berkovich, column 1, lines 63-67).

Further, Berkovich does not disclose or suggest sending the refer request to a requested processor of the at least two processors, as further required by claim 1. As discussed, Berkovich discloses that each PE is connected to MCUs, and that instructions of the program are allocated to PEs to avoid communication with other PEs during execution.

Further, Berkovich does not disclose or suggest returning refer results, by the requested processor referring to the variable, to the refer requestor processor, as further required by claim 1. As discussed, Berkovich does not disclose or suggest that processors that execute executable form programs do any referring or returning of refer results to other processors that execute executable form programs. Accordingly, even taken together in combination, Martin and Berkovich do not disclose or suggest the recitations of claim 1.

APPENDIX

Claim 1. A method of converting a software program comprising object files for a single processor to a software program for a multiprocessor comprising at least two processors, the method comprising the steps of:

allocating each of the object files to at least one of the at least two processors by an object file unit so that the object files are divided into the same number of groups as the number of the at least two processors;

preparing an executable form program for each processor of the at least two processors so that at a time of program execution by a respective processor of the at least two processors each executable form program is running on a single memory space by an operating system on the multiprocessor;

exception processing for a refer requester processor of the at least two processors, by detecting an occurrence of a refer request[[,]] to ~~variables~~ a variable arranged on a memory space managed by another processor of the at least two processors during running of the executable form program;

sending the refer request to a requested processor of the at least two processors;

returning refer results, by the requested processor referring to the ~~variables~~ variable, to the refer requester processor; and

emulation-executing by the refer requester processor a variable refer command from the returned refer results to return to a next command from the exception processing.

Claim 4. The method of converting the software program for the single processor to the software program for the multiprocessor according to claim 1 or 2, wherein the refer request is a write request for writing into the ~~variables~~ variable.